**1a)**

**Bugs:**

* cannot assign \*ptr + 1 = 20
* The while loop would decrease ptr before printing, so it would miss the 0th element
* The while loop would also print the elements backwards instead of starting from the 1st element
* \*ptr + 1 = 20 wouldn’t work because the address of ptr is being dereferenced before 1 is added and so it’d just be arbitrarily adding an int to the value ptr is pointing to

int main(){

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 30; // set arr[0] to 30

ptr++;

\*ptr = 20; // set arr[1] to 20

ptr ++;

ptr[0] = 10; // set arr[2] to 10

ptr --;

ptr--;

for (int i = 0; i < 3; i++){

cout << \*ptr << endl;

ptr++;

}

}

**1b)**

This function won’t set the pToMax parameter to point to the max item because the parameter int\* pToMax is passing the pointer by value and since the function is void, the function won’t change or return anything relevant to the max position.

In order to set the pToMax parameter to the max position, it needs to be passed by reference → int\*& pToMax

void findMax(int arr[], int n, int\*& pToMax)

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

for (int i = 1; i < n; i++)

{

if (arr[i] > \*pToMax)

pToMax = arr + i;

}

}

**1c)**

The value of ptr is uninitialized when it’s entered as a parameter in the call to the computeCube function. In order to initialize ptr to the location of the integer parameter in the function, an int can be declared as 5 and then the ptr should be initialized to the address of that int so that it can be used accordingly in the function.

int main()

{

int x = 5;

int\* ptr = &x;

computeCube(x, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

**1d)**

The code is currently checking the locations of the cstrings against each other and not the elements themselves. Treating cstrings as pointers, str1 and str2 would be the position of the current character being accessed, however, in the while loop as well as the if statement and the return, the code should be checking the characters themselves, not the location. In the return statement, the position of the characters are not going to be the same so that boolean statement would never be true. To check that they’re both 0 at that position, it must be \*str1 and \*str2 instead. In order to access the character at the position instead of the position itself, it must be \*str1 and \*str2 instead, so that the value at the location is being accessed.

bool strequal(const char str1[], const char str2[])

{

while (\*str1 != 0 && \*str2 != 0)

{

if (\*str1 != \*str2) // compare corresponding characters

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

**1e)**

The main function is attempting to access a local array made within the getPtrToArray function through the pointer created in main that was initialized as the call to getPtrToArray(n). Since the array was created and initialized within the function getPtrToArray, the array is a local variable and is temporary; its scope is limited to only the function it was created in. Although it returns a pointer referring to it, there’s no guarantee with what happens to the contents of the array since it’s a local variable. Because the function returns before the printing of the elements in main, accessing the elements of the array that ptr is pointing to in main is thus undefined behavior.

**2a)**

double\* cat;

**2b)**

double mouse[5];

**2c)**

cat = &mouse[4];

**2d)**

\*cat = 25;

**2e)**

\*(mouse + 3) = 54;

**2f)**

cat -= 3;

**2g)**

mouse[2] = 42;

**2h)**

cat[0] = 27;

**2i)**

bool b = (\*cat == \*(cat+1));

**2j)**

bool d = (\*cat == mouse[0]);

**3a)**

double mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

for (int i = 0; i < numScores; i++){ // integer variable

tot += \*(ptr + i);

}

return tot/numScores;

}

**3b)**

const char\* findTheChar(const char\* str, char chr)

{

const char\* ptr = str;

for (int k = 0; \*(ptr+k)!= 0; k++){

if (\*(ptr+k) == chr){

return (ptr + k);

}

}

return nullptr;

}

**3c)**

const char\* findTheChar(char\* str, char chr)

{

while (\*str != 0){

if (\*str == chr) return str;

str ++;

}

return nullptr;

}

**4)**

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 };

int\* ptr = maxwell(array, &array[2]); //the function maxwell returns a pointer to the element with the larger value and so ptr is initialized as location of the larger element between array[0], which is 5, and array[2], which is 4. thus ptr points to the first element of the array because 5 > 4

\*ptr = -1; // value of first element is set to -1 since ptr is pointing to the first element of the array, the array is now {-1, 3, 4, 17, 22, 19}

ptr += 2; // location of ptr is moved +2, thus it now points to the 3rd element of the array

ptr[1] = 9; // ptr[1] would be the element after the 3rd element since ptr is based at position 2 in the array, ptr[1] would be array[3] -- the 4th element; the 4th element is set to 9 so the array is now {-1, 3, 4, 9, 22, 19}

\*(array+1) = 79; // \*(array +1) would be the position after the first element of the array, which is array[1]; array[1] is set to 79 so the array is now {-1, 79, 4, 9, 22, 19}

cout << &array[5] - ptr << endl; // the value being outputted is the difference between the location of array[5] and ptr; since ptr is pointing to the 3rd element of the array and array[5] is the 6th element, 6-3 = 3

This cout line prints 3 and a new line.

swap1(&array[0], &array[1]); // swap1 doesn’t successfully swap any elements of the array because it’s solely swapping the pointers that are passed into the function, not the values themselves, and so the elements of the original array are not being modified

swap2(array, &array[2]); // swap 2 swaps the values of the first element and the 3rd element of the array and so then the array is {4, 79, -1, 9, 22, 19}

for (int i = 0; i < 6; i++)

cout << array[i] << endl; // this cout line outputs the current array, which is {4, 79, -1, 9, 22, 19} , with one element on each line

}

This cout line prints

4

79

-1

9

22

19

**5)**

void removeS (char \* str){

char\* ptr;

while (\*str != 0){ // checking through all chars of str

if (\*str == 's' || \*str == 'S'){

ptr = str;

while (\*(ptr) != 0){

\*ptr = \*(ptr+1);// shift chars

ptr ++;

}

str--; // moving counter back one

}

str ++;

}

}